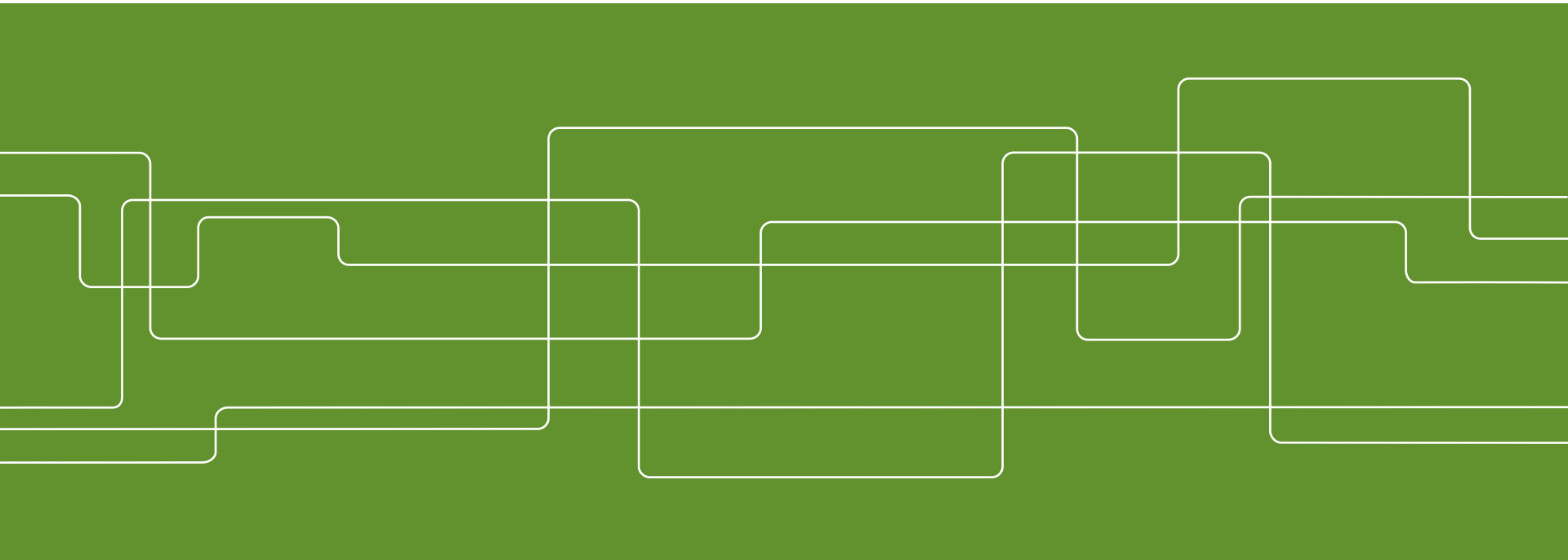
EP1200 Introduction to Computing Systems Engineering

Machine and Assembly Language

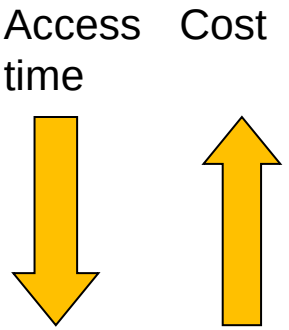


Perspective on sequential logic

Real memory units are highly optimized

- using a great variety of storage technologies

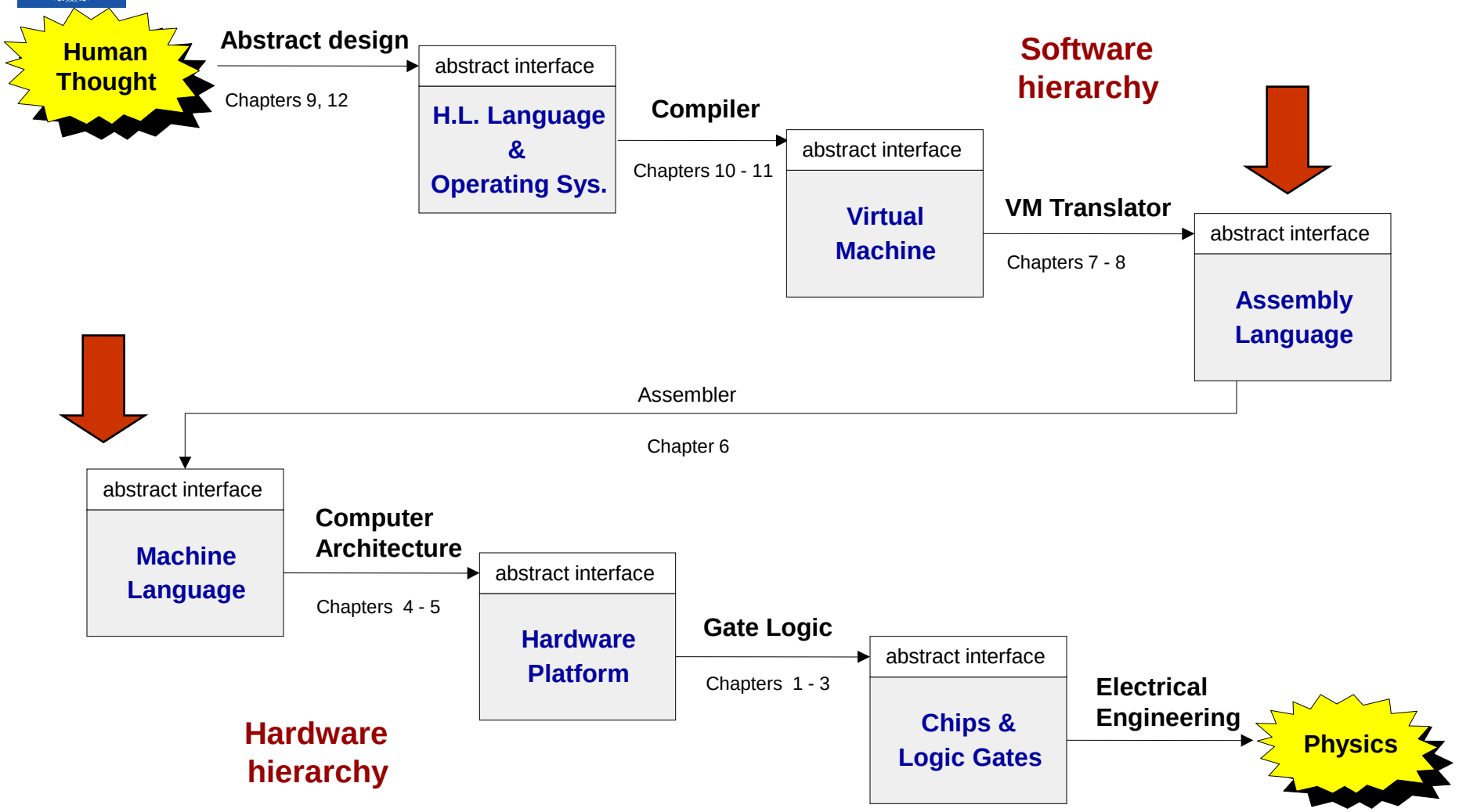
- typical memory hierarchy
  - SRAM (“static”), typically used for the cache
  - DRAM (“dynamic”), typically used for main memory
  - Disk



- volatile (cash, RAM) and non-volatile memory (ROM, disc, flash memory)

Content-addressable Memory
- finds whether given data is stored in the memory
- supplies address or additional information
- used for fast search for example in networking devices

Where we are at:

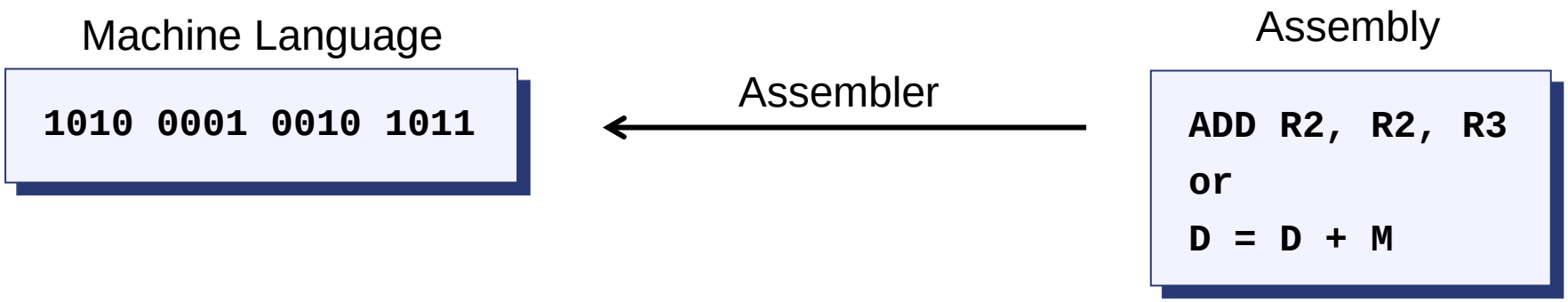


Machine language

- Machine languages: real near hardware programming, to instruct the computer hardware
- Depends on the hardware platform – many versions
- Computer HW typically contains:
  - Processor: performs operations
  - Memory: store program and data
  - Registers: to store address and data for an operation
- Machine language =
  an agreed-upon formalism for manipulating a *memory* using a *processor* and a set of *registers*

Machine language

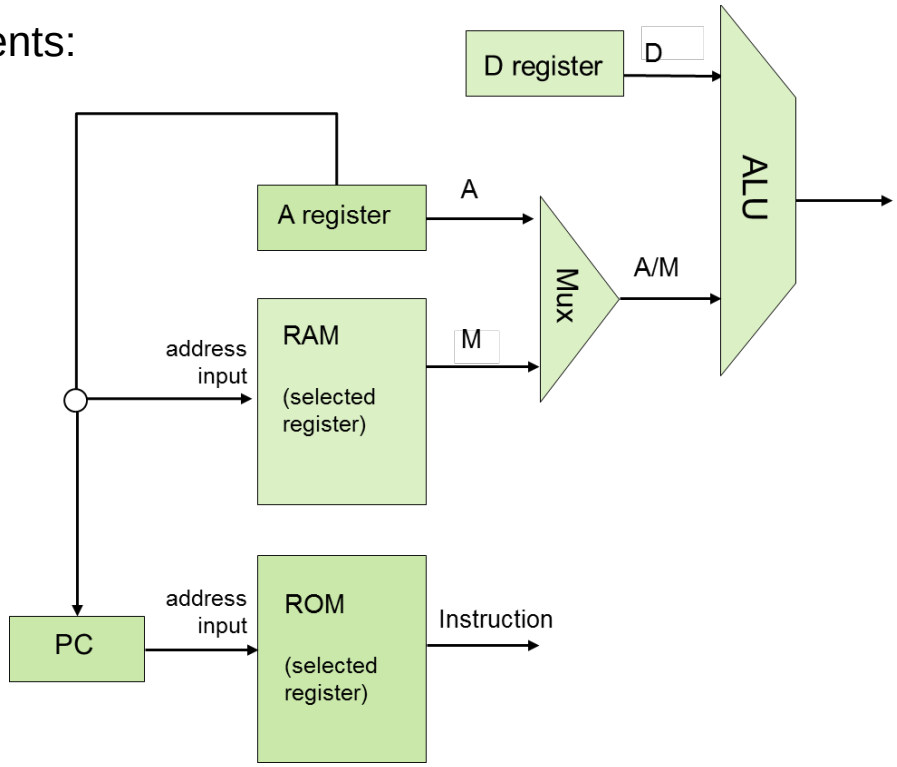
- Binary notation: Machine Language
- Symbolic notation: Assembly
- Requires a translator – Assembler



The Hack computer

We need to design a machine language for the Hack computer

A 16-bit machine consisting of the following elements:



Data memory:
RAM – an addressable sequence of registers

Instruction memory:
ROM – an addressable sequence of registers

Registers:
D, **A, M,** where **M** stands for **RAM[A]**

Processing:
ALU, computing various functions

Program counter:
PC, holding an address for the **ROM**

Control:
The **ROM** is loaded with a sequence of 16-bit instructions, one per memory location, beginning at address 0.

NOTE: all instructions have to be coded with only 16 bits!

The A-instruction and C-instruction

A-instruction: set the A register to a 15 bit value

- value: integer or symbol
- Binary: `0vvv vvvv vvvv vvvv`

C-instruction:

- What to compute? (a,c)
- Where to store? (d)
- What to do next? (j)
- Binary: `111a cccc ccdd djjj`

```
// A-instruction
@value        // A ← value
```

```
// C-instruction
dest=comp;jump
   // dest, jump may be empty
```

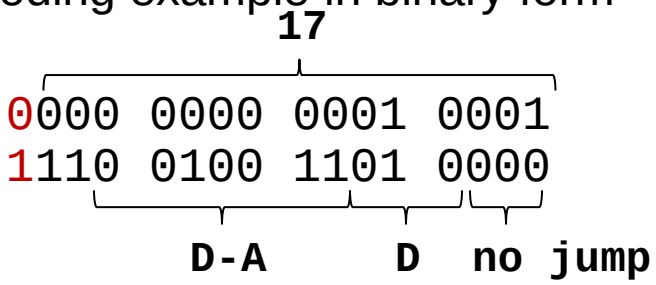
Coding examples:

```
@17    // A = 17
D = D-A  // D = D-17
```

```
@100    // A = 100
D;JEQ   // If D=0 goto 100
```

```
@sum   // sum refers to some
   RAM location
M=0    // sum=0
```

1st coding example in binary form



Complete assembly program example

C language code:

```
// Adds 1+...+100.
 int i = 1;
 int sum = 0;
 while (i <= 100){
    sum += i;
    i++;
 }
```

- Variables: (e.g., i, sum)
- Labels: (e.g.,LOOP)
- Commands: (e.g.,JGT)
- Note: operations with memory access need two instructions.

Hack assembly code:

```
// Adds 1+...+100.
      @i      // i refers to some RAM location
      M=1     // i=1
      @sum    // sum refers to some RAM location
      M=0     // sum=0
(LOOP)
      @i
      D=M     // D = i
      @100
      D=D-A   // D = i - 100
      @END
      D;JGT   // If (i-100) > 0 goto END
      @i
      D=M     // D = i
      @sum
      M=D+M   // sum += i
      @i
      M=M+1   // i++
      @LOOP
      0;JMP   // Got LOOP
 (END)
      @END
      0;JMP   // Infinite loop
```

Symbols and Peripheral Devices

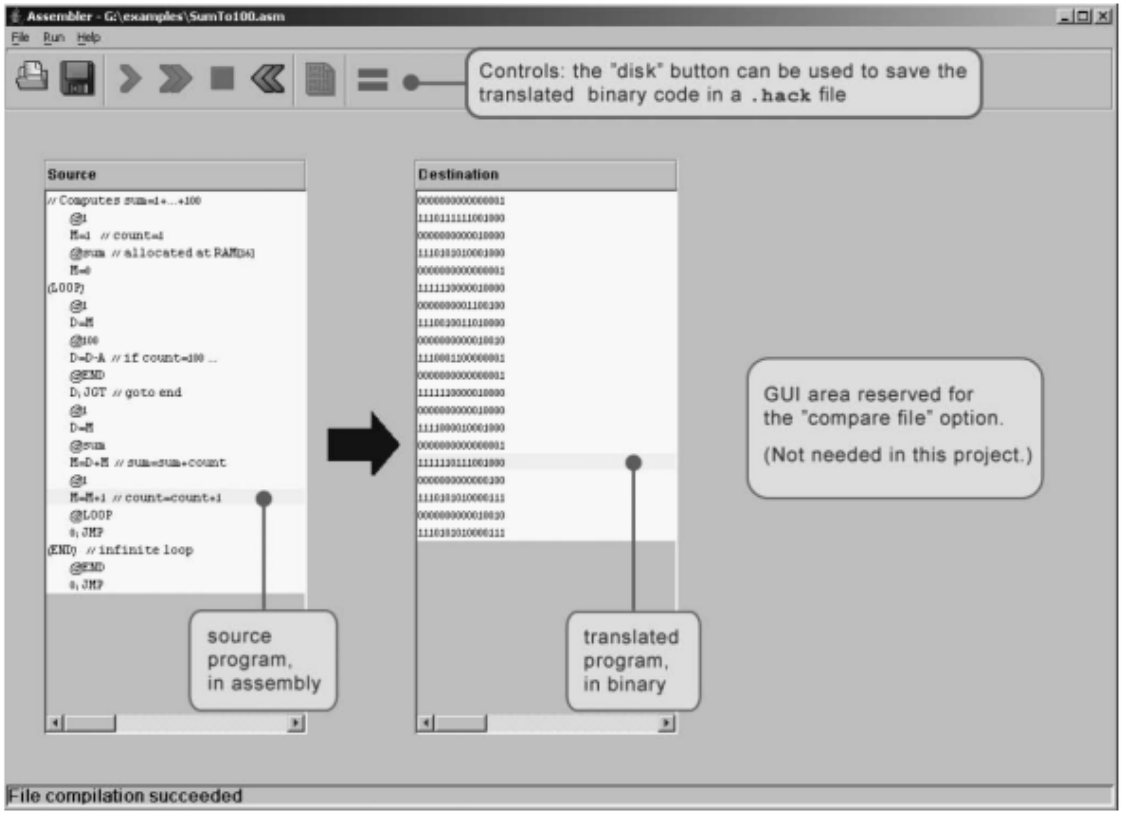
Input/Output

- HACK peripheral devices: screen, keyboard
- Own memory space (memory map), predefined location
- Writing to screen memory map results in black/white pixel
- Reading the keyboard memory means reading which key was pressed

Symbols – to refer to some memory location

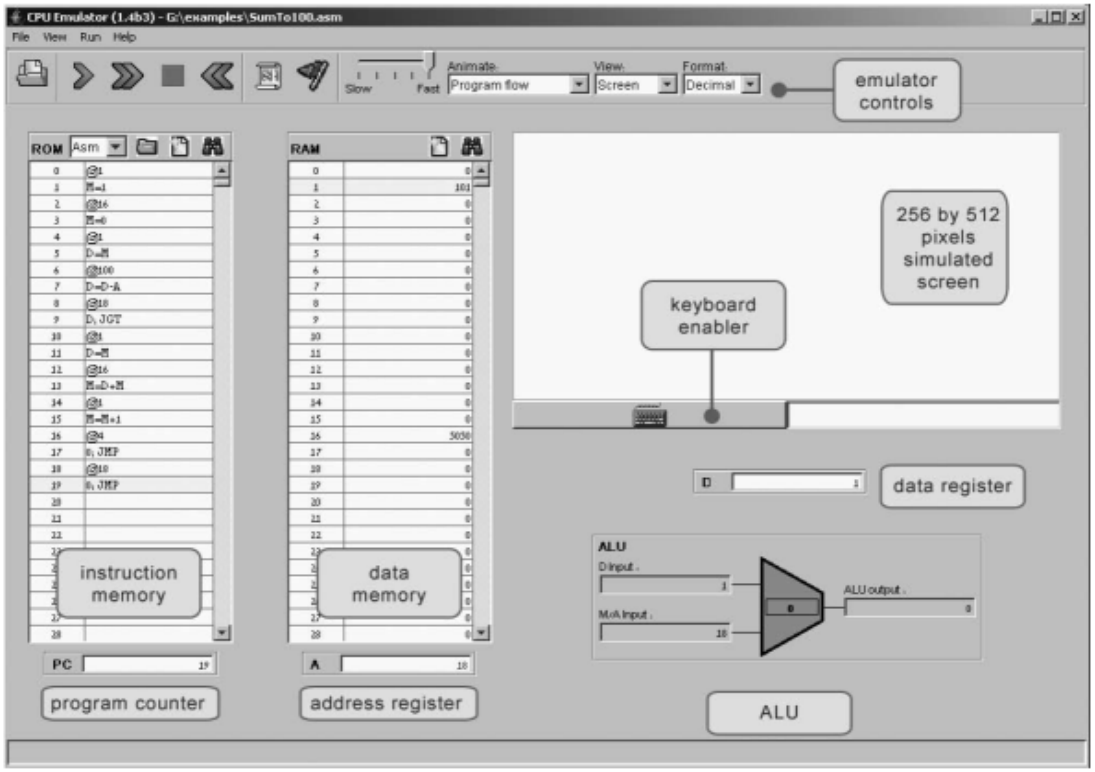
- Predefined in RAM:
  - Virtual registers: R0-R15
  - Predefined pointers: SP (stack pointer), etc for the virtual machine
  - I/O pointers for peripherals: SCREEN, KBD
- User defined: LABELS (ROM location) and variables (RAM location)

Software tools – Assembler



Implements Assembly to Machine language translation

Software tools – CPU emulator



Emulates the CPU (to be built…), runs .hack or .asm

For next class

- Read Chapter 4 Machine Language
- Do project 3 on the course web
  - Mult.asm - First you will need to understand the addition example in the book.
  - Fill.asm – First you may want to experience with writing the screen and reading the keyboard.
- Hand in Project 3 by March 31, 8.00.